MAPPING ROBUST PARALLEL MULTIGRID ALGORITHMS TO SCALABLE MEMORY ARCHITECTURES[1]

Andrea Overman
NASA Langley Research Center
Hampton, VA 23681-0001

John Van Rosendale
ICASE
NASA Langley Research Center
Hampton, VA 23681-0001

SUMMARY

The convergence rate of standard multigrid algorithms degenerates on problems with stretched grids or anisotropic operators. The usual cure for this is the use of line or plane relaxation. However, multigrid algorithms based on line and plane relaxation have limited and awkward parallelism and are quite difficult to map effectively to highly parallel architectures. Newer multigrid algorithms that overcome anisotropy through the use of multiple coarse grids rather than line relaxation are better suited to massively parallel architectures because they require only simple point-relaxation smoothers.

In this paper, we look at the parallel implementation of a V-cycle multiple semicoarsened grid (MSG) algorithm on distributed-memory architectures such as the Intel iPSC/860 and Paragon computers. The MSG algorithms provide two levels of parallelism: parallelism within the relaxation or interpolation on each grid and across the grids on each multigrid level. Both levels of parallelism must be exploited to map these algorithms effectively to parallel architectures. This paper describes a mapping of an MSG algorithm to distributed-memory architectures that demonstrates how both levels of parallelism can be exploited. The result is a robust and effective multigrid algorithm for distributed-memory machines.

[1]This research was supported by the National Aeronautics and Space Administration under NASA contract nos. NAS1-19480 and NAS1-18605 while the second author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

INTRODUCTION

The convergence rate of standard multigrid algorithms degenerates on problems that have anisotropic discrete operators. Such operators arise when the continuous operator is anisotropic or when the discretization is based on highly stretched grids. Although a number of effective cures exist for this difficulty, the best sequential algorithms (based on line or plane relaxation) do not appear to be viable on emerging, massively parallel architectures. Thus, newer algorithms, which achieve robustness through the use of multiple coarse grids rather than line or plane relaxation and require only point-relaxation smoothers, are an attractive alternative.

The problems with line- and plane-relaxation algorithms on parallel architectures have only recently become apparent. Although the tridiagonal systems involved can be solved in parallel by substructured elimination, for example, this approach approximately doubles their computational cost. In addition, a more subtle difficulty exists. The fastest robust sequential algorithms combine line- and plane-relaxation algorithms with semicoarsening. Unfortunately, this means that the size of the line and plane solutions required on coarse grids is the same as on the fine grid. For example, an n^2-point grid in two dimensions with a parallel tridiagonal solver and $O(n^2)$ processors gives a theoretical upper bound on parallel efficiency of only $O(1/\log^2 n)$. Thus, the fact that parallel implementations of such algorithms have proven problematic is not surprising (refs. 1,2,3).

An alternate approach to robustness, based on using multiple grids on every coarse multigrid level, is newer and relatively untried. Through the use of appropriate coarse grids, one can obtain point-relaxation algorithms as robust as line- and plane-relaxation algorithms (refs. 4,5,6,7). However, because of the large number of coarse grids required, these algorithms are not quite competitive with line- and plane-relaxation algorithms on sequential machines. On parallel architectures, the opposite is true (refs. 5,8,9) because the increased parallelism due to the multiple coarse grids is an attractive bonus. In particular, Douglas' method is robust and can be mapped effectively to parallel architectures (ref. 5); Horton (ref. 9) has looked recently at the mapping of Hackbusch's Frequency Decomposition method (ref. 6) to parallel architectures.

In this paper, we study the mapping of the multiple semicoarsened grid (MSG) algorithm, a variant of Mulder's multiple coarse-grid algorithm (ref. 10), to highly parallel architectures. The MSG algorithm (ref. 7) is relatively robust and at the same time provides ample parallelism for current parallel architectures. We take as our model problem the symmetric, positive-definite Helmholtz equation

$$a\,u_{xx} + b\,u_{yy} + c\,u_{zz} - d\,u = f$$

with $a, b, c, d \geq 0$ and focus on the mapping issues involved in implementing this algorithm on distributed-memory architectures such as the Intel iPSC/860 and Paragon.

This paper is organized as follows. We begin with a description of the MSG algorithm in the next section, which is followed by a discussion of observed convergence rates. Our parallel implementation is then described. We present the experimental results, and, finally, conclusions are given.

C-4

We first need to describe the MSG algorithm. For notational simplicity, assume that the domain of the model problem is the unit square in two dimensions and that this problem is to be solved on an $n \times n$ uniform grid as

$$\Omega_h = \{(ih, jh) \mid \quad i = 0, 1, \ldots, n; \quad j = 0, 1, \ldots, n\}$$

with $h = 1/n$. Define the coarser grids $\Omega^{l,m}$, which are obtained by successive semicoarsening of Ω_h l times in the x-direction and m times in the y-direction. Thus, $\Omega^{l,m}$ has $(n+1)/2^l$ grid points in the x-direction and $(n+1)/2^m$ grid points in the y-direction.

Notice that the notation does not distinguish between a grid obtained by semicoarsening first in the y-direction and then in the x-direction and a grid obtained by semicoarsening first in the x-direction and then in the y-direction. Either path leads to a grid of the same shape and size. As shown by Mulder (ref. 10), such equivalent grids must be combined in order to construct reasonable algorithms in three or more dimensions.

Figure 1 shows the interrelations between the various grids for a two-dimensional problem with an 8×8 fine grid. With coarse grids combined as in this diagram, for a 16×16 problem one would have only 16 grids altogether; without combining, the full binary tree of grids would contain 69 grids.

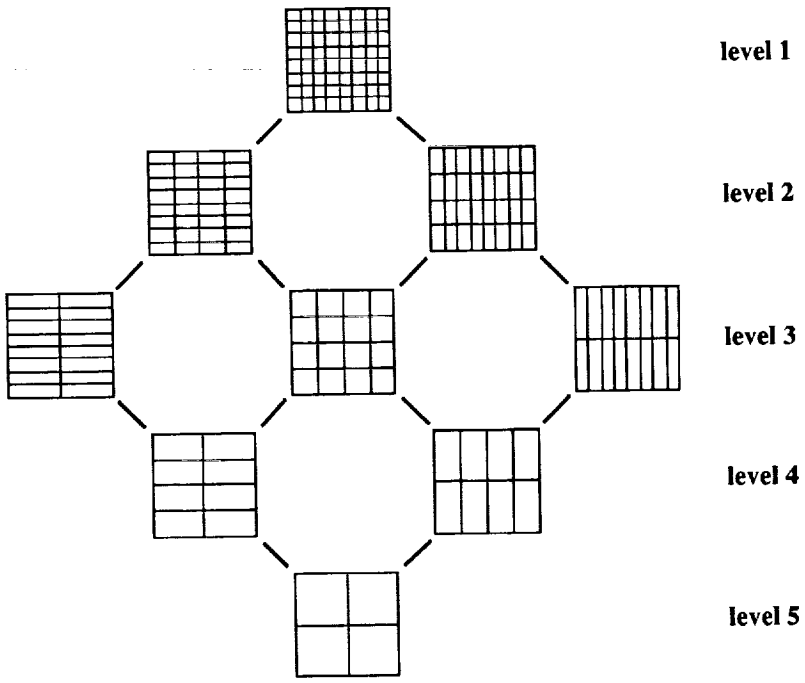

Figure 1. Semicoarsening of an 8×8 grid.

Given this family of grids, one can construct a V-cycle correction scheme analogous to the standard full-coarsening multigrid algorithm. One-dimensional linear interpolation provides a

natural prolongation operator; its adjoint gives the "full weighting" restriction operator. These choices, together with any reasonable smoother, yield a multigrid algorithm. However, the resulting algorithm is not robust.

The problem with this simple correction scheme is explained. If the prolongation is scaled so that the full correction is obtained from the modes that are oscillatory in x but not y and conversely, then the result is double the required correction of the smoothest components that belong to both coarse grids, and divergence results. On the other hand, if the prolongation is scaled to get the proper correction of the smooth components, then some of the oscillatory components are undercorrected, and robustness is lost.

The resolution of this problem is to filter either the residuals that are being restricted or the corrections that are being prolonged to achieve a convergent V-cycle for the model problem

$$a\ u_{xx} + b\ u_{yy}\ =\ f$$

where the convergence rate is independent of $a, b\ >\ 0$. This filtering can be performed in several ways.

Let $v^{l,m}$ denote the correction on grid $\Omega^{l,m}$. Also let R_x and R_y denote restriction in the x- and y-directions, and, similarly, let P_x and P_y denote prolongations. The first effective solution to this problem was given by Mulder (ref. 10). Mulder forms the fine-grid correction

$$P_x\ v^{1,0}\ +\ P_x R_x P_y\ v^{0,1}$$

given solutions $v^{0,1}$ and $v^{1,0}$ on the second level and similar solutions for coarser levels. One can think of the operator $P_x R_x$ here as a high-pass filter that filters out the excess correction for the smooth modes common to both coarse grids.

In recent work, Naik and Van Rosendale have been looking at the analogous scheme with the correction

$$(1\ +\ 1/2\ P_y R_y) P_x\ v^{1,0}\ +\ (1\ +\ 1/2\ P_x R_x) P_y\ v^{0,1}$$

which can be thought of as a symmetric version of Mulder's scheme. A V-cycle proof for one variant of this scheme appears to be possible.

A third way of making the correction is to compute a scalar-valued function $\alpha(x, y)$, which depends on the strength of the discrete differential operator in each coordinate direction. Then, with a properly choosen α, one uses the correction

$$\alpha(x, y) P_x\ v^{1,0}\ +\ [1 - \alpha(x, y)] P_y\ v^{0,1}$$

A V-cycle convergence proof for this scheme, at least for constant coefficient problems, was given in ref. 7. This reference also provides details on the computation of $\alpha(x, y)$.

On sequential machines, any of these schemes is effective and robust. Mulder's scheme and its symmetrized version eliminate the necessity of choosing α; the extra work involved in their interpolations is trivial. However, because the communication required for interpolation is awkward

and expensive on parallel architectures, we used the alpha-switch algorithm here, which reduces the complexity of the interpolations. It is as robust as the alternatives and simpler to implement.

Generalization of this alpha-switch algorithm to the three dimensions is straightforward. Instead of simply computing $\alpha(x,y)$, one computes $\alpha(x,y,z)$ and $\beta(x,y,z)$ and then uses the three weights

$$\alpha(x,y,z) \qquad \beta(x,y,z) \qquad 1 - \alpha(x,y,z) - \beta(x,y,z)$$

From the point of view of parallel architectures, computation of the switching factors α and β is analogous to a Jacobi sweep, which needs to be done only once at the beginning of the computation.

OBSERVED CONVERGENCE RATES

Experimentally, the MSG algorithm converges extremely well for the model problem

$$a\,u_{xx} + b\,u_{yy} + c\,u_{zz} - d\,u = f$$

where the convergence rate is independent of $a,b,c,d \geq 0$ and uniform mesh size. Alternatively, MSG can be used for stretched grids, as shown in Table 1. The results given are observed convergence rates for Poisson's equation with Dirichlet boundary conditions and a random initial guess. Slow variation in the coefficients a,b,c or in mesh spacing have a similar impact on convergence. The Helmholtz term $d \geq 0$ can improve convergence on coarse grids, but is largely irrelevant. All of the above information applies only to problems with smooth coefficients. Special algorithms are required for problems with severe coefficient jumps (refs. 11,3). The discretization used throughout our experiments was a symmetric seven-point finite-difference stencil, with the smoothing done by three red-black successive over-relaxation (SOR) sweeps on every grid.

The problem with this algorithm on sequential machines is the large number of grids required and the resulting high cost per V-cycle. With the usual coarsening by a factor of 2 (as shown in Table 1), the total storage for all grids in three dimensions is eight times that of the finest grid. Thus, the work per V-cycle is also eight times the work on the finest grid, which does not include the cost of the interpolations.

A more attractive sequential algorithm can be made by changing the coarsening factor. In any semicoarsening algorithm, one has fewer Fourier modes to reduce than in full-coarsening algorithms; thus, one can afford to coarsen the grids faster.

If we use coarsening by a factor of 4, for example[2], then the total storage becomes

$$(1 + 1/4 + 1/16 + \ldots)^3 = 64/27$$

times that on the finest grid. Thus, the total work is about $2\frac{1}{3}$ times that on the finest grid.

[2]The red-black SOR smoother used yields poor convergence rates for odd coarsening factors. Thus, the reasonable choices for the coarsening factor are 2 and 4 because either 6 or 8 would make the space of "oscillatory" functions (which must be effectively reduced by the smoother) too large.

Table 1. Convergence Rates of MSG on Various Grids With Factor-of-2 Coarsening

	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$32 \times 32 \times 32$
Uniform Grids			
$dx = 1000, \quad dy = dz = 1$	0.04	0.06	0.07
$dx = 10, \quad dy = dz = 1$	0.04	0.06	0.08
$dx = 0.1, \quad dy = dz = 1$	0.02	0.05	0.07
$dx = 0.001, \quad dy = dz = 1$	0.03	0.07	0.08
Chebyshev Grids			
Chebyshev in x	0.04	0.06	0.11
Chebyshev in x, y	0.04	0.04	0.12
Chebyshev in x, y, z	0.03	0.04	0.15

Table 2 gives the observed convergence rates for the same problems as in Table 1; however, factor-of-4 coarsening was used. Although the convergence rates in Table 2 are poorer than in Table 1, the reduced computational cost per V-cycle more than compensates for this. Three V-cycles of the algorithm can be accomplished with factor-of-4 coarsening for less than the cost of one V-cycle with a factor-of-2 coarsening. With the 32^3 grid, because $0.3^3 = 0.027$, the three V-cycles with a factor-of-4 coarsening are more effective than one V-cycle with a factor-of-2 coarsening.

Massively parallel architectures that have hundreds or thousands of processors might change these considerations and increase the effectiveness of the algorithm with a factor-of-2 coarsening because it provides more parallelism on coarse grids. However, because the algorithm with a factor-of-4 coarsening seemed to provide ample parallelism and the memory per processor is limited on the Intel iPSC/860, we used a factor-of-4 coarsening in our code.

In addition to the use of a factor-of-2 coarsening, the parallelism can be further increased by use of concurrent iteration on all grid levels (refs. 12,13). This form of MSG is particularly attractive on SIMD machines, where the mapping strategies needed for the V-cycle algorithm are prohibitively complex. In joint research with J. Dendy, this alternative is currently being explored for problems with severe coefficient jumps. However, while the concurrent iteration version of MSG maps very nicely to SIMD machines (ref. 7), its convergence rate is in the range of 0.5–0.6, even with a factor-of-2 coarsening. Thus, one trades numerical performance for massive parallelism.

Table 2. Convergence Rates of MSG on Various Grids With Factor-of-4 Coarsening

	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$32 \times 32 \times 32$
Uniform Grids			
$dx = 1000, \quad dy = dz = 1$	0.21	0.20	0.23
$dx = 10, \quad dy = dz = 1$	0.21	0.20	0.24
$dx = 0.1, \quad dy = dz = 1$	0.11	0.13	0.18
$dx = 0.001, \quad dy = dz = 1$	0.11	0.15	0.14
Chebyshev Grids			
Chebyshev in x	0.19	0.18	0.26
Chebyshev in x, y	0.11	0.14	0.25
Chebyshev in x, y, z	0.05	0.19	0.26

MAPPING MSG TO SCALABLE ARCHITECTURES

The V-cycle MSG algorithm achieves fast convergence and contains substantial parallelism, although exploitation of this parallelism is fairly awkward. This awkwardness is in contrast to the standard (full-coarsening) multigrid, where parallel implementation is straightforward. For the MSG case, we designed a program to compute efficient mappings of the algorithm to a distributed-memory architecture. The computed mappings were then implemented with the PARTI[3] runtime primitives developed at ICASE (refs. 14,15). Although this implementation was complex, without PARTI or analogous tools, implementation would have been prohibitively difficult. In this section, we describe our implementation strategy.

Load Balancing

Two basic issues must be addressed in mapping the V-cycle MSG algorithm to distributed-memory architectures: processors must be assigned to the grids on each level and each grid must be partitioned across the processors assigned to it. Because a large number of possible mapping strategies exist, we made two major simplifying choices. First, we chose to map each multigrid level independently of the mapping of all other levels. Second, if the number of processors was greater than the number of grids on a level, we chose to assign each processor to, at most, one

[3]PARTI is an acronym for Parallel Automated Runtime Toolkit at ICASE.

grid on that level.

The first assumption is justified by the observation that the smoothing iteration is more frequent and more computationally intensive than the interpolation, so that the achievement of a good mapping during the smoothing step is crucial to performance. Also, any mapping that achieves an approximate load balance during the smoothing step is bound to induce a large amount of communication during interpolation. One reason for this is that the number of grids on each level almost always differs from the number on neighboring levels; thus, no mapping exists that simultaneously minimizes communication and achieves load balance.

The second assumption that each processor is assigned to no more than one grid on every level was taken to minimize communication, although it does induce some load imbalance. For example, suppose one has three grids on a level to be split over eight processors. Then each grid would ideally receive 2.66 processors. However, such a mapping is complex and clearly increases communication. Instead, one grid would be assigned to two processors, and the other two grids to three each.

In the current implementation, we did not split processors across grids. Instead, we carefully determined those grids that should get fewer and those that should get more processors to achieve approximate load balance without splitting processors across grids. In general, long thin grids (grids with one array dimension much smaller than the others) induce less communication when split over multiple processors than fat grids (grids with all array dimensions about equal). Thus, one maximizes load balance by assigning excess processors to the fattest grids.

Given these preliminaries, our load balancing algorithm follows. By assuming one has p processors and more processors than grids on all multigrid levels, the algorithm for distributing processors to grids is

```
Assign p processors to the finest grid
For level := 2 to max_level {
    ngrids := number_of_grids(level)
    assign ⌊p/ngrids⌋ processors to each grid
    p_excess := p − ngrids ⌊p/ngrids⌋
    assign one more processor to each of the p_excess fattest grids
}
```

We call this the *maximally distributed* strategy.

This algorithm gives a distribution of processors to grids. Afterwards, one still has to partition each grid across the processors. To do this, we blocked the finest grid across processors in all three directions; coarser grids were blocked in one direction. One reason for this choice is that coarser grids often have an odd or prime number of processors, so that partitioning in more than one direction can be quite awkward. In all cases, the direction in which the coarser grids were blocked was chosen to minimize interprocessor communication.

In an alternate implementation referred to as the *aligned* strategy, all coarse grids were aligned to the finest grid, which requires each coarse grid to be partitioned among the full set of processors.

Although this strategy will eliminate communication during the interpolation, it leads to increased communication within a single grid and may quickly lead to idle processors. In the future, a strategy that uses a combination of the two described above may be implemented. In this *hybrid* implementation, coarse grids would be aligned in the first few levels; on lower levels, individual grids would be assigned to only a subset of processors.

PARTI Implementation

As stated, the MSG algorithm was implemented in parallel with the multiblock PARTI routines. The multiblock library was designed to support block-structured aerodynamics codes in which one uses multiple, logically rectangular grid blocks to resolve complex aerodynamic geometries (ref. 16). Because the structure of such codes is fairly similar to that of MSG, we found that the same routines could be effectively used to implement this algorithm.

The PARTI library for block-structured codes allows multiple grid blocks to be processed in parallel and carries out the necessary communication required to move information among the grids. In our parallel implementation that maps coarse grids to subsets of processors, an individual "decomposition" is defined for the fine grid and for each coarser grid. In order to have all processors active on the finest grid, the fine-grid decomposition is embedded into the entire processor space. Then, for each subsequent level, the coarse-grid decompositions are embedded into an approximately equal portion of the processor space, as described in the last section. The single coarse grid on the coarsest level contains few points so it is mapped to one physical processor.

Our parallel version reads a file that holds the grid mapping and distribution information. A subroutine was created to use this mapping information along with the appropriate PARTI routines to set up the problem. As in most multigrid codes, the sequential code uses several large arrays to hold the residual, solution, and right-hand-side data for all grids on all levels. Individual grid sizes and starting index locations into the large arrays are computed and passed as parameters to subroutines. This strategy was maintained in the parallel version; however, the sizes and starting locations were modified to reflect the parallelism and the additional space required for holding boundary data for those grids distributed over more than one processor.

While PARTI aims to require minimal changes to the sequential source program, our parallel implementation was 20 to 25 percent larger than the original sequential program, and some subroutines required an extensive rewrite. Emerging FORTRAN dialects, like High Performance FORTRAN, FORTRAN D, and Vienna FORTRAN, may soon ease this programming burden. However, the current versions of these languages are not expressive enough to allow mapping strategies as complex as those described in this paper. The improvement of such languages, and of software tools like PARTI, is an area of active research at ICASE and elsewhere. The present situation, in which the effective mapping of an algorithm to a parallel architecture is an arduous task of many months, is clearly unacceptable.

EXPERIMENTAL RESULTS

We recently implemented this algorithm and the mapping strategy on a 32-node Intel iPSC/860 and will soon migrate this program to a 64-node Intel Paragon and possibly a CM-5. The current results are preliminary, but are sufficiently encouraging to suggest the relative efficacy of this class of algorithms. For a problem with 16^3 mesh cells, the achieved efficiencies are given in Table 3.

Table 3. Efficiency of Problem With 16^3-Point Grid on iPSC/860

Processors	1	2	4	8	16
Efficiency	1.0	.83	.66	.42	.25

Table 4. MSG Performance on the Intel iPSC/860

Size	Nodes	Total Time (secs)	V-cycle Time, (secs)	
			First V-cycle	Subsequent V-cycles
16^3	1	6.96	3.07	1.22
	2	4.21	1.70	.804
	4	2.63	1.05	.508
	8	2.07	.925	.373
	16	1.71	.793	.302
32^3	4	22.6	11.6	3.55
	8	13.5	7.15	2.03
	16	8.39	4.59	1.23
	32	5.27	2.61	.867
64^3	16	49.5	28.8	6.63
	32	24.1	12.1	3.87

These efficiencies were computed relative to the parallel implementation run on one node. A large amount of overhead can be incurred with the runtime software. For the 16^3 problem, the parallel code run on one processor takes approximately four times longer than the sequential code that contains no PARTI calls. For larger problems, the overhead should become less significant.

Another issue here is the choice of stencil. With the 7-point stencils used, the communication/computation ratio is four times greater than for 27-point stencils, and our efficiencies are correspondingly lower. However, the PARTI library does not currently update the corner ghost points needed for the 27-point stencils, so we were restricted to the use of 7-point stencils. This restriction will be changed in the next release of the library.

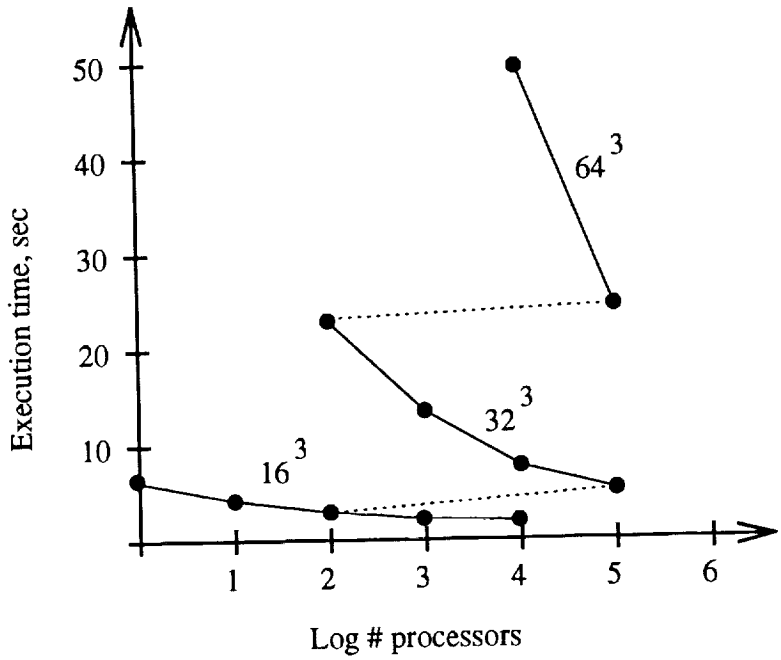

Figure 2. Execution time versus number of processors.

Table 4 shows performance results for several problem sizes. The table contains the overall program timings, along with the timings for each V-cycle. The results show the extra time required in the first V-cycle for setting up the communication schedules. These schedules are saved and, therefore, do not need to be recomputed on subsequent iterations.

Figure 2 expands on the data in Table 4. The graph shows that the 32^3 problem run on 4 nodes requires approximately the same amount of time as the 64^3 problem run on 32 nodes. This result is to be expected because the 64^3 problem has about eight times as much work. In Figure 2, a horizontal connecting line between the two cases (the dashed line on the graph) would indicate the achievement of perfect memory-bounded speedup (ref. 17); however, because of various overheads, this line slopes slightly.

The number of cases plotted here was constrained by current limitations of the PARTI library. For example, we were unable to obtain any timings on the machine that used more than 32 processors. Also, because of the large amount of memory consumed by the PARTI communication library, the user memory available on each processor decreased. These problems should be resolved in future releases of the PARTI library. The multiblock library is in a preliminary stage. We expect that further optimizations will improve the performance of block-structured codes with the multiblock library. The performance effects of some optimizations made to the PARTI primitives used in unstructured codes are described in ref. 18.

Alternate Mapping Strategies

We have also experimented with the *aligned* mapping strategy that was described briefly in

the previous section. With this strategy, the cost of the first V-cycle is much lower than in the *maximally distributed* strategy because the communication that occurs in the interpolation is easier to analyze. However, subsequent V-cycles are more expensive than in the maximally distributed strategy. This difference seems to be due both to the increased communication within each grid (because each grid is subdivided more finely) and to the sequentialization of all grids on every level. As a result, the aligned strategy is less effective than the maximally distributed strategy, even though it reduces interprocessor communication during the interpolation.[4] In future work, we plan to study various hybrid strategies like those proposed in ref. 9 that combine the advantages of both the aligned and maximally distributed strategies.

CONCLUSIONS

We have examined the parallel implementation of a multigrid algorithm based on multiple coarse grids. Such multigrid algorithms have a fast convergence that is independent of grid stretching and can be effectively mapped to highly parallel architectures. We have developed a strategy for mapping such algorithms to parallel machines and have given preliminary results on the effectiveness of this strategy in mapping MSG to the Intel iPSC/860. The PARTI library is being ported to the Intel Paragon; we plan to try our algorithms on this larger machine in the near future.

ACKNOWLEDGMENTS

The authors wish to thank Alan Sussman for making the library available to us while it has been under development and for frequent consultations on the use of the multiblock PARTI routines. We also wish to acknowledge discussions on parallel multigrid issues with Joe Dendy, Naomi Naik, and Graham Horton.

REFERENCES

1. Dendy, J. E., Jr.; Ida, M. P.; and Rutledge J. M.: A Semicoarsening Multigrid Algorithm for SIMD Machines. *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 6, Nov. 1992, pp. 1460-1469.

2. Overman, A.; and Van Rosendale, J.: Mapping Implicit Spectral Methods to Distributed Memory Architectures. ICASE Report 91-52, June 1991.

3. Smith, R. A.; and Weiser, A.: Semicoarsening Multigrid on a Hypercube. *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 6, Nov. 1992, pp. 1314-1329.

[4] A problem also exists with using the PARTI library for the aligned strategy. Currently, PARTI does not handle cases in which the decomposition of a coarse grid across processors results in a processor that receives no mesh points, a case that frequently arises with this strategy. Future versions of the PARTI library may eliminate this restriction.

4. Ta'asan, S.; and Brandt, A.: Multigrid Solutions to Quasi-Elliptic Schemes. In *Progress in Supercomputing in Computational Fluid Dynamics.* E. Murman and S. Abarbanel, eds., Proceedings of the U.S.-Israel Workshop, 1984, pp. 235-255.

5. Douglas, C. C.: A Tupleware Approach to Domain Decomposition Dethods. *Appl. Numer. Math.*, 8, 1991, pp. 353-373.

6. Hackbusch, W.: The Frequency Decomposition Multigrid Method, Part I: Application to Anisotropic Equations. *Numer. Math.*, 56, 1989, pp. 229-245.

7. Naik N.; and Van Rosendale, J.: The Improved Robustness of Multigrid Elliptic Solvers Based on Multiple Semicoarsened Grids. *SIAM J. Numer. Anal.*, vol. 30, no. 1, Feb. 1993, pp. 215-229.

8. Frederickson, P.; and McBryan, O.: Parallel Superconvergent Multigrid. In *Multigrid Methods: Theory, Applications, and Supercomputing.* S. F. McCormick, ed., Marcel Dekker, New York, 1988, pp. 195-210.

9. Bastian, P.; and G. Horton, G.: Parallelization of Robust Multigrid Methods: ILU Factorization and Frequency Decomposition Method. *SIAM J. Sci. Stat. Comput.*, vol. 12, no. 6, Nov. 1991, pp. 1457-1470.

10. Mulder, W.: A New Multigrid Approach to Convection Problems. *J. Comp. Phys.*, vol. 83, 1989, pp. 303-329.

11. Alcouffe, R. E.; Brandt, A.; Dendy, J. E. Jr.; and Painter, J. W.: The Multi-Grid Method for the Diffusion Equation with Strongly Discontinuous Coefficients. *SIAM J. Sci. Stat. Comput.*, vol. 2, 1981, pp. 430-454.

12. Gannon, D.; and Van Rosendale, J.: Highly Parallel Multigrid Solvers for Elliptic PDE's: An Experimental Analysis. ICASE Report 82-36, 1982.

13. Gannon, D.; and Van Rosendale, J.: On the Structure of Parallelism in a Highly Concurrent PDE Solver. *J. Parallel and Distributed Comp.*, vol. 3, 1986, pp. 106-135.

14. Sussman, A.; Saltz, J.; Das, R.; Gupta, S.; Mavriplis, D.; Ponnusamy, R.; and Crowley, K.: PARTI Primitives for Unstructured and Block Structured Problems. *Computing Systems in Engineering*, vol. 3, no. 1-4, 1992, pp. 73-86.

15. Chase, C.; Crowley, K.; Saltz, J.; and Reeves, A.: Parallelization of Irregularly Coupled Regular Meshes. ICASE Report 92-1, Jan. 1992.

16. Vatsa, V.; Sanetrik, M.; and Parlette, E.: Development of a Flexible and Efficient Multigrid-Based Multiblock Flow Solver. AIAA Paper 93-0677, Jan. 1993.

17. Gustafson, J.; Montry, G.; and Benner, R.: Development of Parallel Methods for a 1024-Processor Hypercube. *SIAM J. Sci. Stat. Comput.*, vol. 9, 1988.

18. Das, R.; Mavriplis, D. J.; Saltz, J.; and Gupta, S.: The Design and Implementation of a Parallel Unstructured Euler Solver Using Software Primitives. AIAA Paper 92-0562, Jan. 1992.